**Design Decisions & Reflection**

*Justify development choices for your 3D scene. As you write, think about why you chose your selected objects. Also consider how you were able to program for the required functionality.*

*Explain how a user can navigate your 3D scene. As you compose your thoughts, discuss how you set up to control the virtual camera for your 3D scene using different input devices.*

*Explain the custom functions in your program that you are using to make your code more modular and organized. Ask yourself, what does the function you developed do and how is it reusable?*

The original selection of objects was quick, as I just grabbed whatever objects were on or near my desk. I did try to include as much variety of basic shapes as I could within those objects, but I didn't take a variety of textures into account. Most of the original objects are plastic with a single color which isn't very interesting. Even my desk is almost black, so the plane was not interesting. Therefore, I altered the texture on a couple objects for academic purposes and interest. The basic shapes I decided to use (though some became a little more complex, again for interest and the challenge), were, cube (scaled to be a rectangular prism), cylinder (using 6 and 8 edges), pyramid and plane. These basic shapes were used to represent real 3D objects, but simplified. In addition to shapes and textures, I included 3 lights - a blue point light, a white movable white light, and an orange directional light.

Since I was creating multiple basic shapes whose vertices and indices were hard coded manually, I used a custom "BasicShape" class to create multiple instances of every object I needed. This made it modular and reusable. The class used a CreateShape() method which required the vertices and the indices as vectors, as well as OpenGL's transform matrix and its shader memory location. Each instance kept track of its own position, rotation, and scale and had the ability to update OpenGL's transform later on when the BasicShape's draw() function was called. It made sense to me to give the BasicShape class the responsibility of knowing what its own transform data was. I allowed the class to keep OpenGL's transform memory location since this program only used 1 shader for its objects (there was another shader used purely for visualizing lights as a solid object, but you could give it the light shader on the light's initialization). This gives the OpenGL caller the flexibility to specify which shader should be used for a particular object. I could have included a method to change the shader the object used, but it was not necessary for such a small program. As an afterthought, the Light class was nearly identical to the BasicShape class, but had additional functionality, so I should have had the Light class inherit from the BasicShape class.

Any code that needed to be repeated was placed within a function. I also kept code that didn't need to repeat within OpenGL's Initialize() function and any code that needed to be updated each frame within OpenGL's run() function or within the BasicShape's methods that could be called automatically. For example, BasicShape would need to update OpenGL's active VAO and its data, so the code was placed within BasicShape's draw() method.

Camera input and controls were inspired and driven by the need to debug issues and get a much better view at what was happening in the scene. Its time consuming and frustrating to have to run the program, make a small visual change and re-run it, which is inevitable but can be reduced, and sometimes its easy to misunderstand what an issue is without being able to move and look around. For example, if you run the program and only see a blank screen when you expect to see a triangle, its possible that the data for an object is incorrect on a buffer or that the object is simply translated above or to the side of the camera, or that the camera placement and angle is wrong. Being able to move and look around helps to narrow the range of issues down since perhaps you will see the object slightly above or behind you. To that end, I also included controls to move one of the lights around to inspect lighting such as specular reflection. The controls are as follows:

**Controls**

w move forward

s move backward

a strafe left

d strafe right

e move up

q move down

p toggle perspective / orthogonal mode (flickers if held down)

mousewheel up increase camera speed (increments up to max)

mousewheel down decrease camera speed (decrements down to min)

mouse(pitch/yaw axis) rotate camera (has upper and lower boundary limits)

left arrow move light left (within boundary limits)

right arrow move light right (within boundary limits)

up arrow move light up (within boundary limits)

down arrow move light down (within boundary limits)

1 set draw mode to fill

2 set draw mode to wireframe